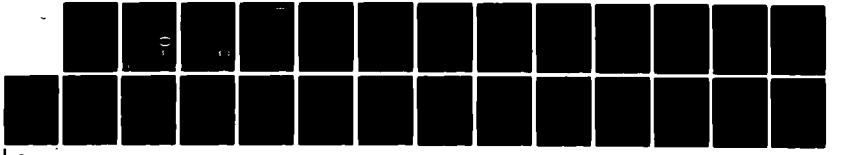
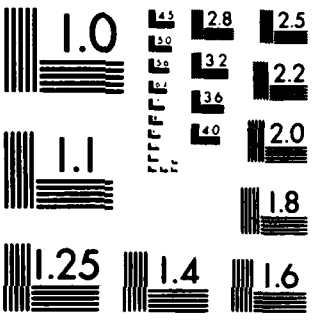
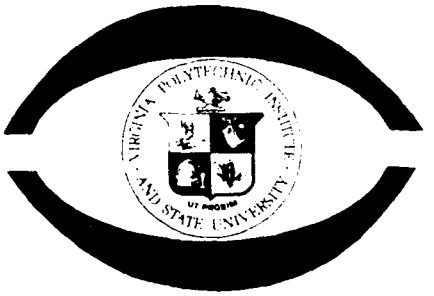
AD-A137 940 A REVISED STONEMAN FOR DISTRIBUTED ADA (TRADEMARK) 1/1
SUPPORT ENVIRONMENTS(U) VIRGINIA POLYTECHNIC INST AND
STATE UNIV BLACKSBURG DEPT OF C.. J P GOODWIN JAN 84
UNCLASSIFIED CS830010 N00014-83-K-0643 F/G 9/2 NL



END
DATE FILMED
3-84
DTIC



Computer Science Department



Virginia Polytechnic Institute
and State University
Blacksburg, Virginia 24061

A REVISED STONEMAN FOR DISTRIBUTED ADA* SUPPORT ENVIRONMENTS

Jeremy P. Goodwin**

Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24061

CS830010
October 10, 1983

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: CS830010	2. GOVT ACCESSION NO.: AD-A137940	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): A REVISED STONEMAN FOR DISTRIBUTED ADA * SUPPORT ENVIRONMENTS		5. TYPE OF REPORT & PERIOD COVERED: Technical
		6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Jeremy P. Goodwin		8. CONTRACT OR GRANT NUMBER(s): N00014-83-K-0643
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Department, VPI & SU, Blacksburg, VA 24061		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Code 442, 800 North Quincy St., Arlington, VA 22217		12. REPORT DATE: January, 1984
		13. NUMBER OF PAGES: 25
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): AFWAL/AAAF, Wright-Patterson AFB, Ohio 45433		15. SECURITY CLASS. (of this report): Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approval for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada, Programming Languages, Programming Environments, APSE, Open Systems Interconnection

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper extends the conceptual model of the 'STONEMAN' document to more completely model the interfaces and protocols that exist in the Ada Programming Support Environment (APSE). A previous extension to the STONEMAN model is reviewed and critiqued, the guidelines for the APSE set forth in STONEMAN are reviewed, and an updated model is proposed. The new model is shown to meet the guidelines set forth in STONEMAN, and to include subsequent ideas as well. The new model is →

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20.
then applied to the problem of user communication with an APSE, and it is shown how the new model extends to include distributed APSEs, and it is shown how the new model extends to include distributed APSEs as well as single host APSEs. The issue of security enforcement, as a necessary subset of dynamic verification, is also included in the new model.

A REVISED STONEMAN FOR DISTRIBUTED ADA* SUPPORT ENVIRONMENTS

Jeremy P. Goodwin**

Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24061

Tech. Rep. No. CS830010
January 13, 1984

ABSTRACT

This paper extends the conceptual model of the "STONEMAN" document to more completely model the interfaces and protocols that exist in the Ada Programming Support Environment (APSE). A previous extension to the STONEMAN model is reviewed and critiqued, the guidelines for the APSE set forth in STONEMAN are reviewed, and an updated model is proposed. The new model is shown to meet the guidelines set forth in STONEMAN, and to include subsequent ideas as well. The new model is then applied to the problem of user communication with an APSE, and it is shown how the new model extends to include distributed APSEs as well as single host APSEs. The issue of security enforcement, as a necessary subset of dynamic validation, is also included in the new model.

---

* Ada is a registered Trademark of the AJPO, U.S. DoD.

** This research was supported by the Ada Joint Program Office through the Office of Naval Research, Information Sciences Division, under ONR contract number N00014-83-K-0643. The effort was monitored by Virginia L. Castor Wright-Patterson AFB, Ohio, and supervised by Dr. J.A.N. Lee, Virginia Tech. Reproduction in whole or in part is permitted for any purpose of the United States Government. The views expressed herein are solely those of the author.

I. INTRODUCTION

A fundamental objective of the Department of Defense (DoD) initiative to develop Ada was to increase the portability and maintainability of embedded software [1]*. To achieve this objective, the Ada Joint Program Office (AJPO) is working to ensure that Ada remains as independent of specific computing systems and applications as possible. The Ada language has been accepted by the American National Standards Institute (ANSI) as a national standard, and has been proposed to the International Organization for Standardisation (ISO) as an international standard. The Ada Validation Organization (AVO) has been established to enforce and protect the trademark for the language. However, the Ada project has evolved beyond an effort toward a common programming language for embedded software systems; work has been begun to define requirements for a common Ada Programming Support Environment (APSE), a Kernal Ada Programming Support Environment (KAPSE), and the Common APSE Interface Set (CAIS). The CAIS is part of the Ada standardization effort because it will provide a standardized development and runtime environment for Ada programs.

A previous report, "Validation in Ada Programming Support Environments" [2], recommended that the Open Systems Interconnection Reference Model be accepted as the underlying model of APSEs, and that there be developed a 'Strawman' to extend

---

* Numbers in brackets refer to references at the end of the report.

Ada systems into a networking environment, based on the OSI Reference Model. The model proposed in [2] will be called a LAPSE (Layered Ada Programming Support Environment) in this paper. That report further suggested that the security aspects of the design of APSEs be investigated and that the results of that study be incorporated into the Stoneman requirements [3]. This paper will examine these ideas further, critiquing the LAPSE and proposing an updated and more detailed model, called the DAPSE (Distributed Ada Programming Support Environment). The DAPSE models communication between APSE tools through the use of the OSI Reference Model but takes into account the need to extend Ada systems into distributed environments. This model also incorporates a security layer, as recommended by [2].

II. REVIEW OF LAPSE MODEL

This section reviews the LAPSE model suggested by [2] (see Figure 1). In that report, the authors noted that "the original intent of the OSI Reference Model was not to actually represent an implementation strategy but instead to model those elements of a communications environment which need attention" (pg. 8). In other words, the OSI model was not intended to force all implementations to have seven layers, but rather to encourage all implementors to layer their implementations, and to clearly specify the functionality of each layer. Thus an application of the OSI model to a specific implementation could quite conceivably merge several layers into one, and split one of the OSI layers

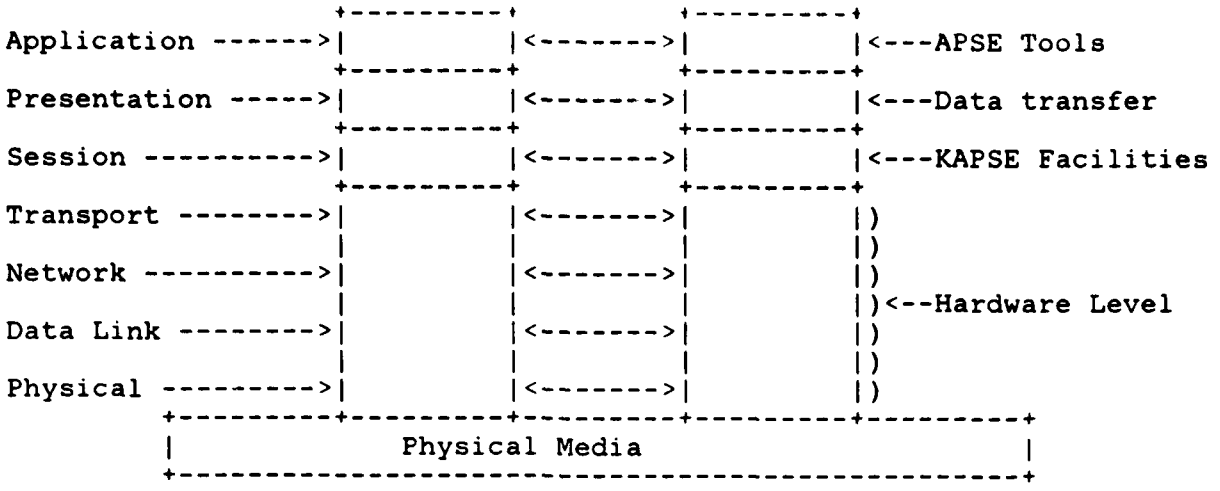


Figure 1. LAPSE Model

into one or more sublayers. This argument was given as a justification for the LAPSE model. The LAPSE model had three layers, which corresponded conceptually to the upper three layers of the OSI Reference Model. The bottom four layers of the OSI model (typically implemented in hardware) were not indigenous to the LAPSE. The top layer of the OSI model (The Application layer) was mapped onto the top layer of the LAPSE model, called the APSE layer. If the current environment was a Minimal Ada Programming Support Environment, (MAPSE), then the top layer will only include the necessary and sufficient toolset, not the user programs or additional tools. Conceptually, there is no difference between an APSE and a MAPSE. The second layer down (corresponding to the OSI Presentation layer) was named the Data Transfer Layer. This layer was to act as an interface layer between the APSE and the KAPSE, and to implement the validation and security mechanisms suggested in the report. The Data Transfer layer accomplished all matching of formal and actual parameters, and associated typechecking. The

bottom layer was the KAPSE Facilities layer, and was mapped onto the Session layer of the OSI model. The report [2] suggested that the KAPSE could be implemented as a collection of Ada packages.

The LAPSE had the advantage that it clearly delineated what interfaces and protocols existed. The report called attention to the existance of certain "hidden protocols" within the STONEMAN model, and noted that these presented a difficult validation problem. In the LAPSE model, these hidden protocols were no longer hidden; they were revealed as KAPSE layer to KAPSE layer communication. Furthermore, the LAPSE model was better than the STONEMAN model because it took into account both validation and security, by modeling the interfaces and protocols that must be validated, and providing a layer where dynamic validation and security checks could be performed. The STONEMAN model was not detailed *enough* to reveal these problems since it was only two dimensional.

III. PROBLEMS WITH LAPSE MODEL

The LAPSE model had three basic problems. The first, and most basic, was that it made an unacceptable use of the OSI model. Since the OSI Reference Model is a model of a communications environment, and not a programming environment, it was inappropriate to fit the Ada Programming Support Environment onto only the upper three layers of the OSI model. Nonetheless, the principles of the OSI model are very much appropriate and ought be adopted in the design of the APSE environment. That is, the APSE

should be layered, and the internal implementation of one layer should be changeable without necessitating a revision or rewriting of code in the other layers. Furthermore, specific functionality should be assigned to each layer of the implementation, and this functionality should follow the overall design principle of layered abstraction, where the services provided by each layer are implemented only in terms of (and by calls to) the functionalities of the layer immediately below.

A second problem with the LAPSE model was that it failed to take into account (or even mention) the Data Base Conceptual Schema [4]. The APSE model should be integrated with the data base model, so that the design of the total environment is consistent and so that the interfaces between the Data Base and the KAPSE are well defined and easy to validate.

The third problem with the LAPSE model, and the problem with the STONEMAN model before it, was that the model was not well enough developed. It did not show where any the validation mechanisms were to be installed. Report [2] stated that all interfaces and protocols must be validated in order to validate the environment, and it was this need that inspired the Data Transfer layer, where validation of the APSE to KAPSE interface was to occur. The problem was that this was not the only place where validation of protocols or interfaces needed to occur. All the horizontal protocols between two instances of layers at the same level must also be validated. For instance, a compiler could be communicating with an editor. Both these programs would reside in the top layer. The compiler and editor would

communicate via a virtual protocol which would be implemented by calls through the interface to the Data Transfer layer. The Data Transfer layer services used by the compiler and the editor would also communicate via a virtual protocol which would be implemented by calls through another interface to services in the KAPSE Facilities layer. The KAPSE Facilities services would communicate using an actual protocol. The LAPSE model did not provide a mechanism whereby the virtual protocols could be validated. Another place where the LAPSE model needed further development was in the distinction between dynamic and static validation. The validation mechanisms suggested in [2] are all static, yet the report suggested the creation of a layer where validation could occur. This validation would be dynamic, and the report did not expand upon the mechanisms by which the validation would be accomplished. It should be noted that there are two types of dynamic validation: dynamic validation during the development cycle, and dynamic validation in the runtime environment. The first type of dynamic validation occurs in the Ada Programming Support Environment on the host machine, and the second type of dynamic validation occurs in the Ada Runtime Environment on the target machine. Both of these sets of validation mechanisms must be considered in any comprehensive APSE model *.

The report [2] also mentioned the need for some security mechanisms, and suggested that they also be implemented in the

---

* Dynamic, built-in "validation" might be better termed "verification"; however, some of the connotations of the term verification are outside the domain of this report and thus the term validation is used consistently.

Data Transfer Layer. However, [2] did not expand on different types of security mechanisms, nor did it distinguish between friendly and non-friendly or security intensive environments in its discussion of the need for security mechanisms. If a program is allowed to operate in an environment where security is important, and that program is not secure or does not meet the security requirements associated with its environment, then the validity of that environment has been jeopardized. Thus two fundamental principles about security must be recognized: (1) Different instances of Ada Programming Support Environments may have different security requirements, and therefore a program that is valid in one instance of the Ada environment may not be valid in another instance of the Ada environment because it is not secure, and (2), security is a subset of validation, i.e. a program that is not secure (enough) is not correct. These ideas clearly need much more attention before a final model for Ada environments can be approached.

Most of these issues were not addressed in the previous report because it was preliminary and did not go into the level of detail that would be necessary to deal with all these issues. This paper attempts to investigate all these issues and refine the model in the light of its findings. Thus the process of successive refinement of the model is carried one more step, arriving at a new model more complete than the earlier model. No doubt further iterations will need to take place.

IV. REQUIREMENTS

Before updating the previous model, the requirements for the APSE should be reiterated and augmented by the ideas that have been put forward since STONEMAN. The basic STONEMAN philosophy [3] emphasized fourteen general guidelines. Five among them are (1) long term software support, (2) host to target system software portability, and (3) an integrated database and toolset, (4) overall simplicity, and (5) uniformity of protocol. The user interface to the APSE programs or tools should not be direct, but should be a virtual interface to the KAPSE with minimal JCL functions such as LOGIN/LOGOUT, CONNECT/RUN and control character processing. Terminal interface drivers should not be part of the KAPSE, but should interface to it, much as the APSE tools do. This model of user communication with the system is similar to one used in many operating systems; user I/O is handled by a special I/O processor (hardware, software or firmware) and buffered into the kernal of the operating system. A second concept that has received attention since STONEMAN is the configuration or version group, made up of shareable objects, each of which has a name and attributes. Each object in the version group has a date as one of its attributes, with later dated objects superceeding earlier ones in subsequent configurations. A third concept, which is related to the second, is that the APSE environment may in practice be distributed, either between the host and target machine (for down-line loading, trace data collection, or emulation) or between several hosts (for resource or information sharing or

communication).

V. DESCRIPTION OF THE DISTRIBUTED APSE (DAPSE) MODEL

The DAPSE model (see figure 2) is a three layer version of model presented in [2]. The top layer is called the APSE layer

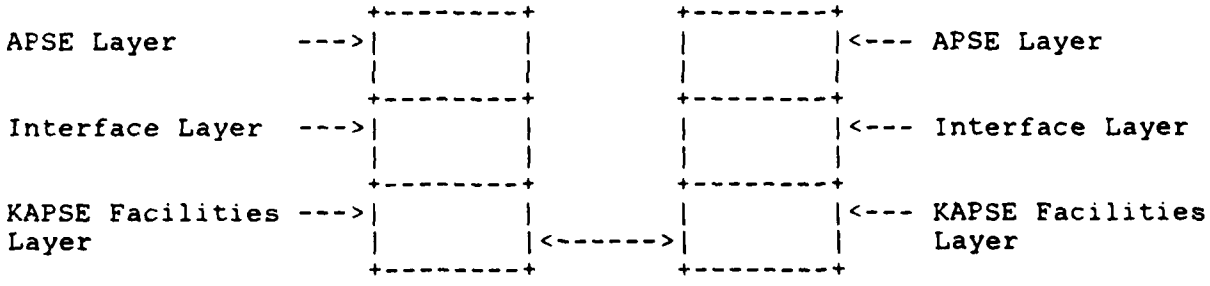


Figure 2. DAPSE Model

since it includes all APSE tools and application programs. It is analogous to, but not overlayed upon, the Application layer of the OSI Reference model. Conceptually, the user visualizes himself at this level, since the programs he interacts with are contained in this level: editors, compilers, debuggers, etc. If this layer contains only the minimal necessary tools and not any other tools or applications programs, then it is a Minimal Ada Programming Support Environment (MAPSE). Thus a MAPSE is a minimal instance of an APSE.

The middle layer is called the Interface layer. Interface is meant in the sense that it has been used in STONEMAN, and this layer has grown out of and is an expansion of the interface line between the APSE and KAPSE in the STONEMAN model. The parameter passing mechanisms that match the APSE actual parameters to the

KAPSE actual parameters and perform dynamic typechecking between them exist in this layer. These mechanisms, formerly represented as the thick line between APSE and KAPSE, have been made into a level due to their complexity. The security mechanisms called for by [2] also exist in this layer and work along with the typechecking mechanisms. The principle is to detect and stop security exceptions at as early a point as possible. This layer dynamically prevents protected KAPSE packages from even being called. Static validation mechanisms can show that the Interface layer properly performs its functionalities, and then use that assertion when validating the KAPSE facilities. The Interface layer performs all dynamic interface validation, including but not restricted to typechecking and package access control. In addition, any necessary data transformation operations may be performed within this layer. This layer can, like the KAPSE, be implemented as a set of Ada packages, with a correspondence between the names of the KAPSE level packages and the Interface level ones. The KAPSE level packages will only be visible from the Interface layer, and the APSE programs must call a KAPSE facility by calling the appropriate Interface layer package. The Interface layer will perform its validation and conversion functions, and then invoke the KAPSE package with the validated and possibly modified argument list.

The bottom layer is the KAPSE layer. It is composed of a set of Ada packages that implement the kernal function of the Ada Programming Support Environment. The KAPSE is the only layer that will be implemented differently on different host machines, but

regardless of implementation its functionalities will be the same over all instances of the APSE. Part of the KAPSE may be implemented in another language, such as the operating system language of the source machine, but as much as possible should be written in Ada itself, since it is validatable. No other layer should have any non-Ada code. The functionality of the KAPSE should be defined in such a way as to provide a general purpose set of operating system type primitives which are robust enough to use to implement the rest of the DAPSE. The functionality of the KAPSE should not be defined in such a way as to prejudice the implementation of the KAPSE toward any one particular architecture among those on which the Ada environment must run.

Although three layers have been defined, this does not preclude each layer from being broken down into sub-layers in implementation. This is, in fact, anticipated as being the natural outgrowth of the development of a system whose underlying model is a layered one.

VI. RATIONALE FOR A LAYERED MODEL

A layered model follows the guidelines set forth by STONEMAN. It aids long term software support by defining and consistently maintaining the functionality of the KAPSE, so that internal modifications to the KAPSE layer will not affect application software, or any other software above the KAPSE layer. Since any program will only reference those programs (or packages) in the layer immediately below its own, and since the functionality (but

not necessarily the implementation) of those programs or packages is fixed across all systems, portability for all software above the KAPSE layer is enforced. Since all protocols and interfaces are clearly defined for the entire system, no hidden protocols exist. By rigidly enforcing one means of inter-tool communication through the KAPSE, an integrated toolset is arrived at and uniformity of protocol is enforced. A layered model is at the same time both simple and complete. For the applications programmer, the APSE is a set of procedures, packages and tasks that are visible to his program. Thus the concepts of the APSE are those of the Ada language itself, and the most straightforward possible. By judiciously grouping the visible packages into a small but well organized set of packages the functionality of the level below can be preorganized for the programmer. Only the necessary portions of the packages need be made visible, and the rest of the underlying layer need not concern the programmer. This model makes the APSE an easy environment to use as a programmer and to maintain as an APSE maintainer, for the same reasons that abstraction makes any system easier to understand and maintain.

VII. RATIONALE FOR THE INTERFACE LAYER

Since the other two layers have been present in both the STONEMAN model and the LAPSE model, there is no need to motivate their presence in the DAPSE model. The Interface layer, however, needs further motivation. The first argument in favor of

including this layer in the model is that this layer was really always present. In STONEMAN, the interface between the KAPSE and the APSE was presumed to perform all of the functionality now assigned to the Interface layer. No new functionality has been added, except for the need for security mechanisms as a subset of the validation mechanisms. The case for the inclusion of security mechanisms somewhere in the model has already been made by [2]. The Interface layer is where these mechanisms belong, along with the other dynamic validation mechanisms. The second argument in favor of the new layer is that the interface mechanisms are too complex, and their effects are too far reaching for them to be ignored in the design of the APSE. If the interfaces are to be designed then they must be visible as a part of the model. They should not be allowed to fall into the crack between the APSE and KAPSE. Thirdly, since validation is a repeated step in the life of all APSEs, it is better to provide for it ahead of time. This was the basic thrust of the recommendation of [2], that validation requirements be established and included in APSE requirements specifications. Finally, since security must be considered to completely validate an APSE system, it is better that the security mechanisms also be designed into the model from the beginning, rather than added after the design has been completed. This is more true of security mechanisms than of others because of the subtle nature of security failures and the low level at which most security mechanisms are implemented.

VIII. DISTRIBUTED ADA PROGRAMMING SUPPORT ENVIRONMENTS

The concept of an APSE as a distributed system or a configured system is also encompassed by the layered model. As in the OSI model itself, the upper layers do not have any knowledge

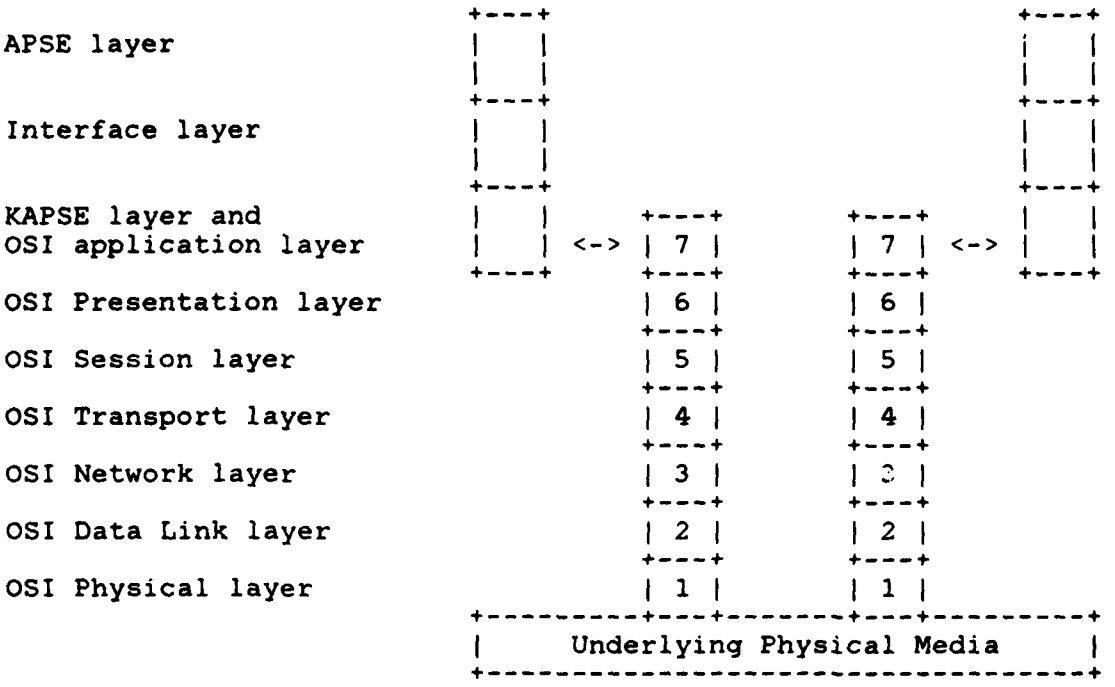


Figure 3. DAPSE model integrated with OSI model

about the physical location of the peer process with which they are communicating. The KAPSE layer to KAPSE layer protocol is the only place where the actual locations of the source and destination programs must be considered. If the source and destination are on the same physical host, then communication may be as modeled above (see figure 2). If not, then the communication is via the network linking the two hosts. It is intended that the networks used to link distributed hosts also be

modeled after the OSI reference model. Thus the model is expanded to the version shown in figure 3 in the case of distributed communication. In fact, only a part of the KAPSE layer, the package specifically concerned with communication, rather than the entire KAPSE layer, need be concerned with whether or not the source and destination are on the same host.

IX. USER COMMUNICATION APPLIED TO DAPSE MODEL

The issue of how a user will interface to the APSE is one that should not be overlooked or put off until late in the design phase if it is to be a natural and consistent interface. The

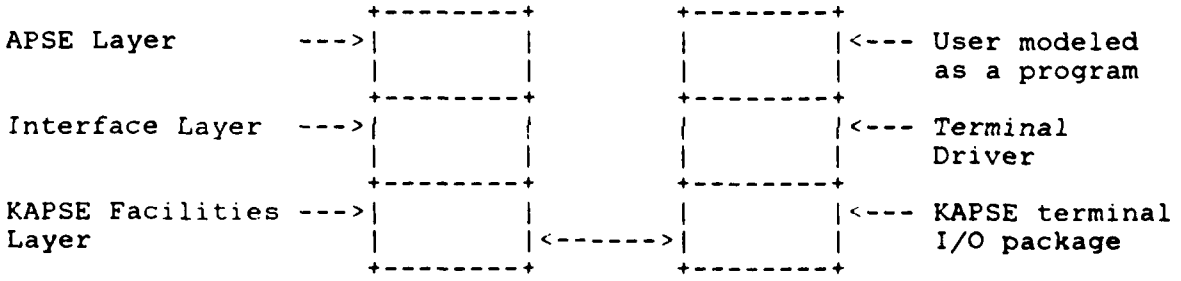


Figure 4. Communication with Users in DAPSE

layered approach handles the user much as if the user were in fact an application program: that is, there is no difference between communication with the user and communication with another APSE program. All communication goes through the KAPSE, and the KAPSE routes the 'message' back up through the appropriate layers to its destination (see figure 4).

X. CONCLUSIONS

The two part model presented in STONEMAN is no longer descriptive enough to model all the considerations that have been added since STONEMAN. A previous report originated the idea that the OSI model be used as the underlying model of APSEs. *It is recommended that a three layered model, patterned after the OSI Reference model, be adopted as the underlying model of APSES.*

This report and a previous one have motivated the need for dynamic validation. *It is recommended that an investigation of dynamic typechecking and validation techniques be performed, and that the results be incorporated into the design of the middle layer of the proposed model.*

Security, as a subset of dynamic validation, is a necessary consideration in any validation system. *It is recommended that a security mechanism be designed and incorporated into the middle layer of the proposed model as early as possible* to enforce the assertion that no KAPSE facility can be called unless the appropriate security test has been passed.

It may be found that the security mechanisms proposed by the research recommended above are difficult or impossible to implement in Ada as it is now defined. *It is recommended that subsequent research be undertaken to investigate ways in which Ada could be extended to include security and validation mechanisms as a part of the language, rather than as a part of the APSE or Ada Runtime Support Environment.*

XI. REFERENCES

[1] Carlson, W.E., Druffel, L.E., Fisher, D.A., and Whitaker, W.A., "Introducing Ada", Proc. 1980 ACM Ann. Conf., ACM, New York NY, 1980, 539 pp.

[2] Kafura, D.E., Lee, J.A.N., Lindquist, T.E, and Probert, T., "Validation in Ada Programming Support Environments", technical report, 1982

[3] Buxton, J.N., Requirements for Ada Programming Support Environments, "STONEMAN", U.S. Dept. of Defense, February 1980, pp. 50.

[4] van Griethuysen, J.J., ed., Concepts and Terminology for the Conceptual Schema and the Information Base, ISO TC97/SC5/WG3, ISO TC97 Computers and Information Processing, March 1982.

TECHNICAL REPORTS DISTRIBUTION LIST

Leader Information Sciences
Engineering Sciences Directorate
Office of Naval Research
800 North Quincy St.
Arlington, VA 22217

Office of Naval Research Resident
Representative, Joseph Henry Building
Room 623
2100 Pennsylvania Avenue, N. W.
Washington, DC 20037

Director, Naval Research Laboratory
ATTN: Code 2627
Washington, DC 20375

Defense Technical Information Center
Building 5, Cameron Station
Alexandria, VA 22314

V. L. Castor
AFWAL/AAAF
Wright-Patterson AFB, Ohio 45433

Dr. Jack Kramer
I.D.A.
1801 N. Beauregard St.
Alexandria, VA 22311

Lt. Cdr Brian Schaar
Ada Joint Program Office
3D139 (400AN)
Pentagon
Washington, DC 20301

Mitch Bassman
Computer Sciences Corp.
6565 Arlington Blvd.
Falls Church, VA 22046
M/C 281

Frank Belz
TRW DSG
One Space Park
R2/1127
Redondo Beach, CA 92078

Tom Conrad
NUSC
Bldg. 1171
Newport, RI 02840

Bob Converse
NAVSEA
PMS-408
Washington, DC 20362

Jay Ferguson
DoD
ATTN: T303, Jay Ferguson
9800 Savage Rd.
Ft. Meade, MD 20755

Jack Foidl
TRW DSG
3420 Kenyon Street *202
San Diego, CA 92110

John Foreman
Texas Instruments, Inc.
P. O. Box 405 M/S 3407
Lewisville, TX 75067

Barbara Fromhold
U.S. Army CECOM
DRSEL-TCS-ADA-3
Ft. Monmouth, NJ 07703

Tim Harrison
Texas Instruments, Inc.
P. O. Box 405 M/S 3407
Lewisville, TX 75067

Hal Hart
TRW DSG
One Space Park
R2/1127
Redondo Beach, CA 92078

Doug Johnson
SoftWrights Inc.
1401 N. Central Expressway
Suite 100
Richardson, TX 75080

Larry Johnston
NADC
Code 503
Warminster, PA 18974

Elizabeth Kean
RADC/COES
Griffiss AFB, NY 13441

Rudolph Krutar
Elizabeth Wald
NRL
Code 5150
4555 Overlook Ave., SW
Washington, DC 20375

Bill Laplant
HQ USAF/SITT
Washington, DC 20330

Larry Lindley
NAC D/072.21
6000 E. 21st St.
Indianapolis, IN 46218

Warren Loper
NOSC
Code 8315
San Diego, CA 92152

Lucas M. Maglieri
National Defense Hqds.
101 Colonel Bay Dr.
Ottawa, Ontario K1A0K2

Jo Miller
NWC
Code 3192
China Lake, CA 93555

Gil Myers
NOSC
Code 8322
San Diego, CA 92152

Philip Myers
Dave Pasterchik
NAVELEX
ELEX 8141A
Washington, DC 20360

MITRE Corp.
K203
P. O. Box 208
Bedford, MA 01730

Eldred Nelson
TRW DSG
One Space Park
R2/1076
Redondo Beach, CA 90278

Tricia Oberndorf
NOSC
Code 8322
San Diego, CA 92152

Shirley Peele
Guy Taylor
FCDSSA
Code 822
Bldg. 1279 Dam Neck
Virginia Beach, VA 23461

Lee Purrier
George Robertson
FCDSSA
Code 822
200 Catalina Blvd.
San Diego, CA 92147

Mo Stein
Ed Dudash
NSWC/DL
Code N31
Dahlgren, VA 22448

Tucker Taft
Jim Moloney
Intermetrics
733 Concord Ave.
Cambridge, MA 02138

Rich Thall
SofTech
460 Totten Pond Road
Waltham, MA 02154

Chuck Waltrip
Johns Hopkins University
Applied Physics Lab
Johns Hopkins Road
Laurel, MD 20707

Bill Wilder
SofTech
Three Skyline Place
Suite 500
5201 Leesburg Pike
Falls Church, VA 22041

Bernie Abrams
Charles Mooney
Grumman Aerospace
Mail Station B38-35
Bethpage, NY 11714

Dennis Cornhill
John Beane
Honeywell/SRC
2600 Ridgeway Pkwy.
MN17-2351
Minneapolis, MN 55413

Fred Cox
EES/SEL/DSD
Georgia Tech
Atlanta, GA 30332

Dick Drake
IBM
Federal Systems Division
102/075
Godwin Drive
Manassas, VA 22110

Jon Fellows
System Development Corp.
5151 Camino Ruiz, 02-B14
Camarillo, CA 93010

Herman Fischer
Litton Data Systems
MS 64-30
8000 Woodley Ave.
Van Nuys, CA 91409

Roy Freedman
Hazeltine Corp.
Research Laboratories
Greenlawn, NY 11740

Anthony Gargaro
Computer Sciences Corp.
304 W. Route 38
Moorestown, NJ 08057

Steve Glaseman
Teledyne Systems Co.
19601 Nordhoff St.
Northridge, CA 91324

Eric Griesheimer
Nicholas Baker
McDonnell Douglas Astronautics
5301 Bolsa M/S 11-2
Huntington Beach, CA 92647

Ron Johnson
Boeing Aerospace Co.
3903 Hampton Way
Kent, WA 98032

Judy Kerner
Norden Systems M/S M171
P. O. Box 5300
Norwalk, CT 06856

Reed Kotler
Lockheed Missiles & Space
1111 Lockheed Way
Sunnyvale, CA 94086

Pekka Lahtinen
Oy Softplan AB
P. O. Box 209
SF-33100 Tampere 10
Finland

Eli J. Lamb
Bell Labs - 3A405
600 Mountain Avenue
Murray Hill, NJ 07974

Tim Lindquist
Department of Computer Science
VPI & SU
Blacksburg, VA 24061

Dave Loveman
Massachusetts Computer
Assoc., Inc.
26 Princess St.
Wakefield, MA 01880

Tim Lyons
Software Sciences Ltd.
Abbey House
Farnborough Hampshire
GU14 7NB
England

Dave McGonagle
GE OR&D K-1
Schenectady, NY 12345

H. R. Morse
Frey Federal Systems
Chestnut Hill Rd.
Amherst, NH 03031

Erhard Ploedereder
c/o Tartan Laboratories
477 Melwood Ave.
Pittsburgh, PA 15213

Ann Reedy
PRC
1500 Planning Res Dr.
SW1
McLean, VA 22102

Jim Ruby
Hughes Aircraft Co.
P. O. Box 3310, 618/P215
Fullerton, CA 92634

Sabina Saib
General Research Corp.
P. O. Box 6770
Santa Barbara, CA 93111

Edgar Sibley
Alpha Omega Group, Inc.
World Building Suite 406
8121 Georgia Avenue
Silver Spring, MD 20910

Thomas Standish
Programming Environment
Project - Computer Science Dept.
University of California
Irvine, CA 92717

Rob Westermann
TNO-IBBC
P. O. Box 9
2600 AA Delft
The Netherlands

Herb Willman
Raytheon Company - MSD
Hartwell Road (GRA-1)
Bedford, MA 01730

Doug Wrege
Dianna Humphrey
Control Data Corp.
5500 Interstate N. Pkwy.
Atlanta, GA 30328

Larry Yelowitz
Ford Aerospace & Communications Corp. WDL
3939 Fabian Way MSV02
Palo Alto, CA 94303